

On the Complexity of CCG Parsing

Marco Kuhlmann*
Linköping University

Giorgio Satta**
University of Padua

Peter Jonsson†
Linköping University

We study the parsing complexity of Combinatory Categorical Grammar (CCG) in the formalism of [Vijay-Shanker and Weir \(1994\)](#). As our main result, we prove that any parsing algorithm for this formalism will necessarily take exponential time when the size of the grammar, and not only the length of the input sentence, is included in the analysis. This result sets the formalism of [Vijay-Shanker and Weir \(1994\)](#) apart from weakly equivalent formalisms such as Tree-Adjoining Grammar (TAG), for which parsing can be performed in time polynomial in the combined size of grammar and input sentence. Our proof highlights important differences between the formalism of [Vijay-Shanker and Weir \(1994\)](#) and contemporary incarnations of CCG.

1. Introduction

Combinatory Categorical Grammar (CCG; ([Steedman and Baldridge 2011](#))) is a well-established grammatical framework that has supported a large amount of work both on linguistic analysis and natural language processing. Under the linguistics perspective, the two most prominent features of CCG are its tight coupling of syntax and semantic information, and its capability to encode this information entirely within the lexicon. Despite the strong lexicalization that characterizes this formalism, CCG is still able to use a very compact representation of the lexicon, and at the same time to handle non-local dependencies in a simple and effective way. After the release of CCG-annotated datasets ([Hockenmaier and Steedman 2007](#)), there has also been a surge of interest in this formalism within statistical natural language processing, and a wide range of applications including data-driven parsing ([Clark and Curran 2007](#); [Zhang and Clark 2011](#)), broad-coverage semantic parsing ([Lewis and Steedman 2014](#); [Lee, Lewis, and Zettlemoyer 2016](#)), and machine translation ([Lewis and Steedman 2013](#)).

* Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden.
E-mail: marco.kuhlmann@liu.se

** Department of Information Engineering, University of Padua, via Gradenigo 6/A, 35131 Padova, Italy.
E-mail: satta@dei.unipd.it

† Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden.
E-mail: peter.jonsson@liu.se

Given that CCG is so well-established, it is quite remarkable that there is relatively little formal work on it. The most well-known exception is the seminal article of [Vijay-Shanker and Weir \(1994\)](#), who define a formalism that we shall refer to by the acronym VW-CCG (after its creators) and present a proof of the weak equivalence of this formalism and Tree-Adjoining Grammar (TAG; [Joshi 1985](#)). This equivalence is a milestone of the literature on the so-called mildly context-sensitive grammar formalisms. However, we find it worth pointing out that VW-CCG is rather different from CCG as it is in use today. In particular, the weak equivalence with TAG crucially depends on a feature of this formalism that is not available in modern versions of CCG: the ability to restrict the use of combinatory rules on a per-grammar basis. Without this feature, the generative capacity of VW-CCG is strictly smaller than that of TAG ([Kuhlmann, Koller, and Satta 2015](#)). As we shall see in this article, the availability of rule restrictions also affects the computational complexity of VW-CCG.

Our point of departure is the result by [Vijay-Shanker and Weir \(1993\)](#) and [Kuhlmann and Satta \(2014\)](#), who present a polynomial-time parsing algorithm for VW-CCG. The asymptotic complexity of both algorithms is in $\mathcal{O}(n^6)$, where n is the length of the input sentence. This matches the asymptotic complexity of practical parsing algorithms for TAG, and establishes a nice parallelism to the weak equivalence result mentioned above. However, while the runtime of both parsing algorithms for VW-CCG is polynomial in the length of the input sentence, it is *exponential* in the size of the grammar. This is in contrast with the scenario for TAG, where the runtime of standard parsing algorithms is (roughly) quadratic with respect to grammar size ([Schabes 1990](#)), and it raises the question, left open by [Kuhlmann and Satta \(2014\)](#), of whether the exponential factor is just a property of the existing parsing algorithms, or whether VW-CCG parsing is inherently more complex than TAG parsing when grammar size is taken into account. Our main result in this article is that it is the second alternative. More specifically, we show that *any* parsing algorithm for VW-CCG will *necessarily* take exponential time when the size of the grammar is included in the analysis.

Formally, we study the **universal recognition problem** for VW-CCG. The universal recognition problem for a class of grammars \mathcal{G} is the decision problem defined as follows: Given as input a grammar G in \mathcal{G} and a string w over the alphabet of G , decide whether w is in $L(G)$, the language generated by G . The computational complexity of this problem is measured as a function of the combined size of G and w . The universal recognition problem should be contrasted with the membership problem for any specific grammar G in \mathcal{G} , whose complexity is measured as a function solely of the length of w . The complexity of the universal recognition problem is generally higher than the complexity of the membership problem. For instance, the universal recognition problem for context-free grammars is PTIME-complete (complete for problems solvable in deterministic polynomial time), whereas the membership problem for the same class defines the class LOGCFL, which is generally conjectured to be a proper subset of PTIME.

The definitions of the universal recognition problem and the membership problem often generate some confusion. For instance, in applications such as parsing or translation, we work with a fixed grammar and process a stream of sentences, so it might seem that the universal recognition problem is of no relevance at all in practice. However,

the computational analysis that one can provide when considering the grammar as part of the input almost always reveals how structural components of the grammar itself contribute to the complexity of the parsing task. More precisely, when investigating the universal recognition problem one expresses the computational complexity of string recognition in terms of several parameters, other than the input string length, each indicating individual components of the working grammar, as for instance the number of nonterminals, the maximum size of rules, the maximum length of unary derivations, etc. This provides a much more fine-grained picture than the one that we get when analysing the membership problem, disclosing the effects that each individual feature of the grammar has on string recognition. The same type of analysis is also very useful in detecting and isolating features that are the source of undesired complexity for the formalism at hand, or in the design of grammar formalisms, when one has to deal with the trade-off between expressivity and computational complexity.

The remainder of this article is structured as follows. After presenting the VW-CCG formalism in Section 2, we first study in Section 3 the universal recognition problem for a restricted class of VW-CCG, where each category is “lexicalized.” We show that for this subclass universal recognition is already NP-complete. This implies our main result that parsing algorithms for VW-CCG will take exponential time in the combined size of the grammar and the input string, *under the assumption that* $\text{PTIME} \neq \text{NP}$. In Section 4 we analyze the general case and show that the universal recognition problem for unrestricted VW-CCG is EXPTIME-complete, a stronger statement than the one in Section 3 as it does not make any assumptions at all. We anticipate that many readers will be content with the result in Section 3, especially since the proofs of the more general result in Section 4 are considerably more complex. Finally, Section 5 is devoted to a general discussion of our result, its ramifications, and its relevance for current research.

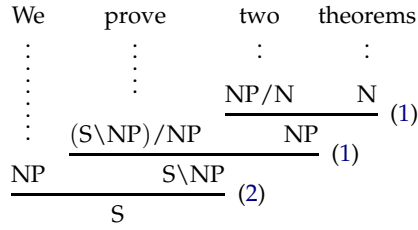
2. Preliminaries

In this section we present the VW-CCG formalism. We assume the reader to be already familiar with the basic notions of categorial grammar, and in particular with the idea of categories as syntactic types. Like other categorial formalisms, a VW-CCG grammar has two central components: a **lexicon**, which specifies the categories for individual words, and a set of **rules**, which specify how to derive the categories of longer phrases from the categories of their constituent parts.

2.1 Lexicon

The VW-CCG lexicon is a set of pairs $\sigma := X$, where σ is a word (formalized as a symbol from some finite vocabulary) and X is a category. Formally, the set of **categories** over a given set \mathcal{A} is the smallest set $\mathcal{C}(\mathcal{A})$ such that (i) $\mathcal{A} \subseteq \mathcal{C}(\mathcal{A})$ and (ii) if $X \in \mathcal{C}(\mathcal{A})$ and $Y \in \mathcal{C}(\mathcal{A})$ then $X/Y \in \mathcal{C}(\mathcal{A})$ and $X \backslash Y \in \mathcal{C}(\mathcal{A})$. Categories of form (i) are called **atomic**, those of form (ii) are called **complex**.

Categories as Stacks. Categories are usually viewed as directed versions of the function types in the simply-typed lambda calculus. Here we find it useful to deviate from this and view categories as *stacks*, as explained in what follows. We treat slashes as left-

**Figure 1**

A sample derivation tree for the sentence “We prove two theorems”.

associative operators and omit unnecessary parentheses. This allows us to write every category $X \in \mathcal{C}(\mathcal{A})$ in the form

$$X = A|_1X_1 \cdots |_mX_m$$

where $m \geq 0$, $A \in \mathcal{A}$ is an atomic category that we call the **target** of X , and the $|_iX_i$ are slash–category pairs that we call the **arguments** of X . Based on this notation we view X as a pair consisting of the target A and a stack whose elements are the arguments of X , with the argument $|_mX_m$ at the top of the stack. Note that arguments can in general contain complex categories.

Example 1

In the derivation shown in Figure 1, lexicon assignments are indicated by dotted lines: The verb *prove* for example is associated with the complex category $(S \backslash NP)/NP$. In our notation, the same category can also be written as $S \backslash NP/NP$. The target of this category is S , and the stack consists of the two arguments $\backslash NP$ and $/NP$, with $/NP$ at the top of the stack. Note that we follow the standard convention in CCG and draw derivations with the leaves at the top and the root at the bottom.

2.2 Rules

The rules of VW-CCG are derived from **combinatory schemata**:

$$\begin{array}{ll}
 \text{X/Y} \quad Y|_1Z_1 \cdots |_dZ_d \Rightarrow X|_1Z_1 \cdots |_dZ_d & \text{(forward schema)} \\
 Y|_1Z_1 \cdots |_dZ_d \quad \text{X}\backslash\text{Z} \Rightarrow X|_1Z_1 \cdots |_dZ_d & \text{(backward schema)}
 \end{array}$$

where $d \geq 0$, the $|_i$ are slashes (forward or backward), and X, Y and the Z_i are variables ranging over categories. Each schema specifies how to combine two input categories, one primary input category (highlighted) and one secondary input category, to produce an output category. The integer d is called the **degree** of the schema. Rules derived from the schemata where $d = 0$ are called **application** rules; the others are called **composition** rules.

Example 2

Figure 2 shows all (six) combinatory schemata with degree at most 1, together with their

$X/Y \ Y \Rightarrow X$	(forward application)	(1)
$Y \ X \backslash Y \Rightarrow X$	(backward application)	(2)
$X/Y \ Y/Z \Rightarrow X/Z$	(forward harmonic composition)	(3)
$X/Y \ Y \backslash Z \Rightarrow X \backslash Z$	(forward crossed composition)	(4)
$Y \backslash Z \ X \backslash Y \Rightarrow X \backslash Z$	(backward harmonic composition)	(5)
$Y/Z \ X \backslash Y \Rightarrow X/Z$	(backward crossed composition)	(6)

Figure 2

The combinatory schemata with degree 0 (application; top) and 1 (composition; bottom).

conventional names. In the derivation shown in Figure 1, each branching of the tree is annotated with the schema used in that step.

A **combinatory rule** over a set of categories $\mathcal{C}(\mathcal{A})$ is obtained from a combinatory schema by optionally restricting the ranges of some of the variables. Two types of restrictions are possible: (i) We may require the variable Y or any of the Z_i to take the value of some specific category in $\mathcal{C}(\mathcal{A})$. For example, we could derive a restricted version of backward crossed composition that applies only if $Y = S \backslash NP$:

$$(S \backslash NP) \backslash Z \ X \backslash (S \backslash NP) \Rightarrow X \backslash Z \quad (7)$$

(ii) We may restrict the range of the variable X to categories with a specific target $A \in \mathcal{A}$. For example, we could restrict backward crossed composition to apply only in situations where the target of X is S , the category of complete sentences. We denote the resulting rule using the “box” notation of Kuhlmann, Koller, and Satta (2015):

$$Y \backslash Z \ S \square \backslash Y \Rightarrow S \square \backslash Z \quad (8)$$

In this notation, the symbol \square is used as a variable for the part of the category stack below the topmost stack element.

Example 3

Backward crossed composition (6) can be used for the analysis of heavy NP shift in sentences such as *Kahn blocked skillfully a powerful shot by Rivaldo* (example from (Baldrige 2002)). A derivation for this sentence is shown in the upper half of Figure 3. However, the schema cannot be universally active in English, as this would cause the grammar to also accept strings such as **Kahn blocked skillfully a powerful by Rivaldo shot*, which is witnessed by the derivation in the lower half of Figure 3 (a dagger † marks the problematic step). To rule out this derivation, instead of the unrestricted schema, a VW-CCG grammar of English may select only certain instances of this schema as rules, in particular instances that combine the two restrictions in (7) and (8). In this way the unwanted derivation in Figure 3 can be blocked, while the other derivation is still admissible. Other syntactic phenomena require other grammar-specific restrictions, including

the complete ban of certain combinatory schemata (cf. (Steedman 2000, Sections 4.2.1–4.2.2)).

$$(S \backslash NP) / NP \quad (S \backslash NP) \backslash (S \backslash NP) \Rightarrow (S \backslash NP) / NP \quad N / N \quad N \backslash N \Rightarrow N / N$$

2.3 Grammars

Derivations. Derivations of G are represented as binary trees whose nodes are labeled with either lexicon entries (leaves) or categories (inner nodes). In order to represent such trees by linear terms, we use the following notation. Let A be some unspecified alphabet.

For $a \in A$, the term a represents a tree with a single node labeled by a . For tree terms t_1, \dots, t_q , $q \geq 1$, the term $a(t_1, \dots, t_q)$ represents the tree whose root node is labeled by a and has q children, which are the root nodes of the trees represented by t_1, \dots, t_q . With this notation, we define the set of **derivation trees** of G and the associated mappings top (which returns the category at the root node of the tree) and $yield$ (which returns the left-to-right concatenation of the symbols at the leaves) recursively as follows:

- Every lexicon entry $\sigma := X$ forms a (single-node) derivation tree τ . We define $top(\tau) = X$ and $yield(\tau) = \sigma$.
- Let τ_L and τ_R be derivation trees with $top(\tau_L) = X_L$, $yield(\tau_L) = w_L$, $top(\tau_R) = X_R$, and $yield(\tau_R) = w_R$, and let $X_L X_R \Rightarrow X$ be a ground instance of some combinatory rule in R . Then $\tau = X(X_L, X_R)$ is a derivation tree. We define $top(\tau) = X$ and $yield(\tau) = w_L w_R$, where juxtaposition denotes string concatenation.

The connection between this formal definition and the graphical notation for derivation trees that we have used in Figure 1 and Figure 3 should be clear. The only difference is that in a formal derivation tree, leaf nodes correspond to lexicon entry $\sigma := X$, while in our graphical notation, leaf nodes are split into a parent node with the category X and a child, leaf node with the symbol σ .

Generated language. The grammar G generates a string w if there exists a derivation tree whose root node is labeled with the distinguished category S and whose yield equals w . The **language** generated by G , denoted by $L(G)$, is the set of all strings generated by G .

Example 4

Vijay-Shanker and Weir (1994) construct the following VW-CCG G (Example 3.3). We only specify the lexicon and the set of rules; the vocabulary, set of atomic categories and distinguished atomic category are left implicit. The lexicon is defined as follows:

$$a := A, \quad b := B, \quad \varepsilon := S \backslash A / \hat{S} / B, \quad \varepsilon := \hat{S} \backslash A / \hat{S} / B, \quad \varepsilon := S, \quad \varepsilon := \hat{S}$$

The set of rules consists of all instances of application and all instances of forward composition of degree at most 3 where the target of the secondary input category is restricted to one of the “hatted” categories. We write Y for a variable restricted to the set $\{S, A, B\}$, \hat{Y} for a variable restricted to the set $\{\hat{S}, \hat{A}, \hat{B}\}$, and Z_i for an unrestricted variable. As before, the $|_i$ are slashes (forward or backward).

$$X/Y \quad Y \Rightarrow X \tag{10}$$

$$Y \quad X \backslash Y \Rightarrow X \tag{11}$$

$$X/\hat{Y} \quad \hat{Y}|_1 Z_1 \cdots |_d Z_d \Rightarrow X|_1 Z_1 \cdots |_d Z_d \quad \text{where } 0 \leq d \leq 3 \tag{12}$$

As witnessed by the derivations in Figure 4, the language generated by this grammar contains the subsets $\{a^n b^n \mid n \geq 0\}$ and $\{(ab)^n \mid n \geq 0\}$.¹

¹ Vijay-Shanker and Weir (1994) are mistaken in claiming that this grammar generates *exactly* $\{a^n b^n \mid n \geq 0\}$.

The ability to impose restrictions on the applicability of rules plays an important role in VW-CCG. As already mentioned in the introduction, VW-CCG is weakly equivalent to Tree-Adjoining Grammar (TAG) (Joshi and Schabes 1997), but this result crucially rests on the availability of rule restrictions (Kuhlmann, Koller, and Satta 2015).

The ability of VW-CCG to assign lexicon entries to the empty string contradicts one of the central principles of CCG, the Principle of Adjacency, by which combinatory rules may only apply to entities that are phonologically realized (Steedman 2000, p. 54). In this section we therefore first investigate the complexity of the universal recognition problem for a restricted version of VW-CCG where this feature is dropped and every lexical category is projected by an overt word. We say that a grammar G whose lexicon does not contain any assignments of the form $\varepsilon := X$ is ε -free.

The universal recognition problem for ε -free VW-CCG is NP-complete.

8

3.1 NP-Hardness

Our hardness proof is by a reduction from the Boolean Satisfiability Problem (SAT). An instance of SAT is given by a Boolean formula ϕ in conjunctive normal form. This means that ϕ is a conjunction of clauses c_i , where each clause consists of a disjunction of one or more literals. A literal is either a variable v_j or a negated variable $\overline{v_j}$. The question asked about ϕ is whether there is a truth assignment to the variables that makes ϕ evaluate to **1** (true). In our reduction we construct an ε -free grammar G and an input string w such that ϕ is satisfiable if and only if $w \in L(G)$, and such that the combined size of G and w is polynomial in the total number of literals in ϕ . We thus obtain the following:

Lemma 1

The universal recognition problem for ε -free VW-CCG is NP-hard.

We start with the construction of the input string w . We then set up the lexicon and the rules of the grammar G in such a way that every derivation for w consists of three clearly separated parts. We present these parts in sequence, introducing the relevant lexicon entries and rules as we go along. The vocabulary and the set of atomic categories will be implicit. Throughout the construction we write m for the number of clauses in ϕ and n for the total number of distinct variables in ϕ . The index i will always range over values from 1 to m (clauses), and the index j will range over values from 1 to n (variables). We illustrate our construction using the following instance of SAT:

$$\phi = (v_1 \vee \overline{v_2}) \wedge (v_1 \vee v_2) \wedge (\overline{v_1} \vee \overline{v_2})$$

For this instance we have $m = 3$ and $n = 2$. We can verify that the only truth assignment satisfying ϕ is $\{v_1 \mapsto \mathbf{1}, v_2 \mapsto \mathbf{0}\}$. We set $c_1 = (v_1 \vee \overline{v_2})$, $c_2 = (v_1 \vee v_2)$, and $c_3 = (\overline{v_1} \vee \overline{v_2})$.

3.1.1 Input String. We construct the input string as

$$w = c_m \cdots c_1 c_0 v_1 \cdots v_n v_{n+1} d_n \cdots d_1$$

where the c_i and v_j are symbols representing the clauses and variables of the input formula ϕ , respectively. The symbols c_0 and v_{n+1} as well as the d_j are special symbols that we use for technical reasons, as explained below. For our running example we have $w = c_3 c_2 c_1 c_0 v_1 v_2 v_3 d_2 d_1$.

3.1.2 Guessing a Truth Assignment. The first part of a derivation for w “guesses” a truth assignment for the variables in ϕ by assigning a complex category to the substring $c_0 v_1 \cdots v_n v_{n+1}$. Figure 5 shows how this could look like for our running example. Reading from the leaves to the root, for every symbol v_j in w , the derivation nondeterministically chooses between two lexicon entries, $v_j := [\$/[v_j \mapsto \mathbf{1}]/\$]$ and $v_j := [\$/[v_j \mapsto \mathbf{0}]/\$]$; these entries represent the two possible truth assignments to the variable. Note that we use square brackets to denote atomic categories. The derivation then uses compositions (13) and (14) to “push” these variable-specific categories to the argument

$$\begin{array}{cccc}
c_0 & v_1 & v_2 & v_3 \\
\vdots & \vdots & \vdots & \vdots \\
\frac{[c_0]/[\$] \quad [\$]/[v_1 \mapsto \mathbf{1}]/[\$]}{[c_0]/[v_1 \mapsto \mathbf{1}]/[\$]} & (13) & \frac{[\$]/[v_2 \mapsto \mathbf{0}]/[\$]}{[\$]/[v_2 \mapsto \mathbf{0}]/[\$]} & (14) \\
\frac{[c_0]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]/[\$]}{[c_0]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}]} & & & \frac{[\$]}{[\$]} & (15)
\end{array}$$

Figure 5

Derivation fragment that “guesses” the truth assignment for the running example.

stack of the lexical category for the special symbol c_0 , and a final application (15) to yield a complex category that encodes the complete assignment.

Lexicon Entries and Rules. To support derivations such as the one in Figure 5, we introduce the following lexicon entries:

$$c_0 := [c_0]/[\$] \quad v_j := [\$]/[v_j \mapsto \mathbf{1}]/[\$] \quad v_j := [\$]/[v_j \mapsto \mathbf{0}]/[\$] \quad v_{n+1} := [\$]$$

We also introduce rules according to the following restricted schemata. Schemata (13) and (14) yield composition rules of degree 2; schema (15) yields application rules.

$$X/[\$] \quad [\$]/[v_j \mapsto \mathbf{1}]/[\$] \Rightarrow X/[v_j \mapsto \mathbf{1}]/[\$] \quad (13)$$

$$X/[\$] \quad [\$]/[v_j \mapsto \mathbf{0}]/[\$] \Rightarrow X/[v_j \mapsto \mathbf{0}]/[\$] \quad (14)$$

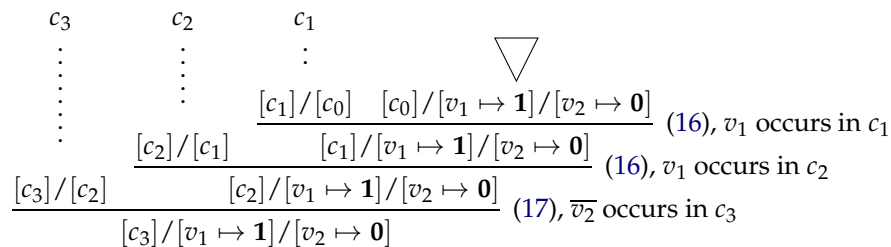
$$X/[\$] \quad [\$] \Rightarrow X \quad (15)$$

Spurious Ambiguity. It is worth mentioning here that our rules support other derivation orders than the left-branching order shown in Figure 5. In particular, we could first combine the variable-specific categories with each other, and then combine the result with the category for c_0 . One could rule out this spurious ambiguity by restricting the target of the primary input of each of the rules above to the category $[c_0]$, obtaining rules such as the following:

$$[c_0] \square / [\$] \quad [\$]/[v_j \mapsto \mathbf{1}]/[\$] \Rightarrow [c_0] \square / [v_j \mapsto \mathbf{1}]/[\$] \quad (13')$$

However, for the purposes of this reduction, the different derivation orders are irrelevant, and we therefore abstain from using target restrictions.

3.1.3 Verifying the Truth Assignment. The second part of a derivation for w verifies that the truth assignment hypothesized in the first part satisfies all clauses. It does so by using compositions to “pass” the stack of atomic categories encoding the truth assignment from one clause to the next, right-to-left. For the running example, this could be done as in Figure 6. Crucially, the rules used in this part are restricted in such a way that the assignment can be “passed” to the next clause c_i only if c_i is satisfied by att



Derivation fragment that verifies the assignment for the running example. The white triangle represents the derivation shown in Figure 5.

Lexicon Entries and Rules. To implement the second part of the derivation, for each clause c_i we introduce a lexicon entry $c_i := [c_i]/[c_{i-1}]$. Our rules make crucial use of variable restrictions. To introduce them we define the following notational shorthands:

$$\begin{aligned}\mathbf{1}_j &\equiv /Y_1 \cdots /Y_{j-1}/[v_j \mapsto \mathbf{1}]/Y_{j+1} \cdots /Y_n \\ \mathbf{0}_j &\equiv /Y_1 \cdots /Y_{j-1}/[v_j \mapsto \mathbf{0}]/Y_{j+1} \cdots /Y_n\end{aligned}$$

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{1}_j \Rightarrow X\mathbf{1}_j \quad \text{if } v_j \text{ occurs in } c_i \quad (16)$$

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{0}_j \Rightarrow X\mathbf{0}_j \quad \text{if } \overline{v_j} \text{ occurs in } c_i \quad (17)$$

Spurious Ambiguity. Similarly to what we noted for the first part (Section 3.1.2), the derivation of this part can proceed in several ways, because at each step we may be able to choose more than one rule to satisfy a clause c_i . For example, in the derivation in Figure 6, instead of using the rule of schema (16) with witness “ v_1 occurs in c_2 ” we could also have used the rule of schema (17) with witness “ $\overline{v_2}$ occurs in c_2 .” However, also as before, there is no need to eliminate this spurious ambiguity for the purposes of this reduction.

$$\begin{array}{c}
\begin{array}{ccc}
& \triangle & \\
[c_3]/[v_1 \mapsto \mathbf{1}]/[v_2 \mapsto \mathbf{0}] & [v_2 \mapsto \mathbf{0}] & \\
\hline
[c_3]/[v_1 \mapsto \mathbf{1}] & & (19)
\end{array}
\end{array}
\begin{array}{c}
d_2 \\
\vdots \\
d_1 \\
\vdots \\
[v_1 \mapsto \mathbf{1}]
\end{array}
\begin{array}{c}
\hline
[c_3]
\end{array}
\begin{array}{c}
(18)
\end{array}$$

Figure 7

Final derivation fragment for the running example. The white triangle represents the derivation shown in Figure 6.

3.1.4 Finalizing the Derivation. The third and final part of a derivation of w reduces the complex category encoding the truth assignment to the distinguished category of G , which we define as c_m , by a sequence of applications. For the running example, this is illustrated in Figure 7.

Lexicon Entries and Rules. This part of the derivation requires two lexicon entries for each of the auxiliary symbols: $d_j := [v_j \mapsto \mathbf{1}]$ and $d_j := [v_j \mapsto \mathbf{0}]$. The rules are:

$$X/[v_j \mapsto \mathbf{1}] \quad [v_j \mapsto \mathbf{1}] \Rightarrow X \quad (18)$$

$$X/[v_j \mapsto \mathbf{0}] \quad [v_j \mapsto \mathbf{0}] \Rightarrow X \quad (19)$$

3.1.5 Time Complexity. We now analyze the time complexity of our reduction. For a given clause c_i , let $|c_i|$ be the number of literals in c_i . We define $|\phi| = \sum_i |c_i|$. The number of rules added in the first and the third parts of the construction of G is in $\mathcal{O}(n)$, and the size of each such rule is bounded by a constant that does not depend on $|\phi|$. For the second part of the construction of G , for each clause c_i we add a number of rules that is at most $|c_i|$, and possibly less if there are repeated occurrences of some literal in c_i . Thus the total number of rules added in this part is in $\mathcal{O}(|\phi|)$. Each such rule has size in $\mathcal{O}(n)$. Putting everything together, and observing that n is in $\mathcal{O}(|\phi|)$, we see that the size of G is in $\mathcal{O}(|\phi|^2)$. It is not difficult to see then that our reduction can be carried out in time polynomial in the size of ϕ .

3.2 Membership in NP

We now turn to the membership part of our proof.

Lemma 2

The universal recognition problem for ε -free VW-CCG is in NP.

For the proof of this lemma, we provide a polynomial-time nondeterministic algorithm that accepts an ε -free VW-CCG G and a string w if and only if G can derive w . We adopt the usual proof strategy where we first guess a derivation tree for w and then verify that this tree is valid.

Size of a Derivation Tree. We need to argue that the total number of characters needed to encode a derivation tree is polynomial in the combined size of G and w . Note that this involves both the tree structure itself and the lexicon entries and categories at the nodes of the tree. We start by observing that any derivation tree with ℓ leaf nodes (labeled with lexicon entries) has exactly $\ell - 1$ binary nodes (labeled with categories). Let τ be an arbitrary derivation tree for w . Since in G there are no lexical categories for unpronounced words, there is a one-to-one correspondence between the leaf nodes of τ and the symbols in w , which implies that the number of nodes of τ is exactly $2|w| - 1$.

Maximal Size of a Category. Now that we have bounded the number of nodes of τ , we will bound the size of the categories that these nodes are labeled with. Consider therefore an internal node v of τ and its associated category X . In order to state an upper bound for $|X|$, we distinguish two cases: If v is a unary node, then $|X|$ is bounded by the largest size of a category in the lexicon of the grammar. We denote this quantity by λ . If v is a binary node, let $X = A|Y_1 \cdots |Y_q$, with A an atomic category. A rule of the grammar can increase the number of arguments of its primary category by at most d , where d is the maximum degree of a rule in the grammar. Let γ be the maximum number of arguments in a category in the lexicon. Since no more than $|w| - 1$ rules are used in τ , we conclude that q is bounded by $\gamma + d(|w| - 1)$. By a lemma of [Vijay-Shanker and Weir \(1994, Lemma 3.1\)](#), every argument Y_i in X must also occur as an argument in some category in the lexicon of G . Thus the size of each argument of X is bounded by the largest size of an argument appearing in a category in the lexicon, a quantity that we denote by α . Putting everything together, we have that $|X|$ is bounded by $1 + \alpha(\gamma + d(|w| - 1))$. From this it is not difficult to see that the overall space needed to encode our derivation tree τ for w along with all of the categories at its nodes is $\mathcal{O}((\lambda + \alpha\gamma)|w| + \alpha d|w|^2)$. This is a polynomial in the combined size of the grammar and the input string.

Nondeterministic Algorithm. We can now provide our nondeterministic algorithm for testing whether G derives w . In a first step we write down a guess for a derivation tree τ for w . Given our space bound on τ , we can carry out this step in time polynomial in the size of G and w . In a second step we visit each internal node v of τ and read its associated category X . If v is a unary node, we check whether X is a lexicon entry for the word at v 's child. If v is a binary node, we check whether X can be obtained by some rule of the grammar applied to the categories at the two children of v . We accept if every check is successful. Even this second step can be carried out in time polynomial in the size of G and w . This concludes the proof of Lemma 2.

3.3 Discussion

In the previous sections we have shown that the universal recognition problem for ε -free VW-CCG is NP-complete (Theorem 1). The result is in contrast with the fact that, for the weakly equivalent TAG formalism, the universal recognition problem can be solved in polynomial time. This naturally raises the question what features of the VW-CCG formalism are the source of this additional complexity. We discuss this question on the basis of the reduction in our proof of Lemma 1. In this reduction we use a combination

of three central features of VW-CCG, listed below. Dropping any of these features would break our reduction.

Lexical Ambiguity. The first feature of VW-CCG that we exploit in our construction is the ability to assign more than one category to some lexical items. In part 1 of the reduction (Section 3.1.2), this allows us to “guess” arbitrary truth assignments for the variables in the clause ϕ . However, the possibility to write grammars with lexical ambiguity is an essential feature of all interesting formalisms for natural language syntax, including also TAG. Therefore, at least in isolation, this feature does not seem to be able to explain the complexity of the universal recognition problem for VW-CCG. Even if our goal was to design a novel version of VW-CCG which can be parsed in polynomial time in the size of the input grammar, we would not seriously consider giving up with lexical ambiguity.

Unbounded Degree of Composition. The second feature of VW-CCG that we rely on is the availability of composition rules without a constant (with respect to the full class of grammars) bound on their degree. This feature is crucial for our encoding of truth assignments. In particular, without it we would not be able to percolate arbitrarily large truth assignments through derivation trees; our construction would work only for formulas with a bounded number of variables.

Unbounded degree of composition certainly seems to be a very powerful feature. Interestingly, some authors stipulate a bound on that degree; for English for example, [Steedman \(2000, p. 42\)](#) puts this bound at $d \leq 3$. More generally, one could ask whether some language-independent constant exists that allows the modeling of natural language syntax by VW-CCG grammars whose maximal degree of composition is bounded by that constant. For any such degree-restricted subclass of grammars, our proof would break, and it may be possible (though not obvious) to devise a polynomial-time algorithm for the universal recognition problem.

Rule Restrictions. The third feature of VW-CCG that we exploit is its ability to put grammar-specific restrictions on combinatory rules. In particular, in part 2 of our construction (Section 3.1.3), we use rules whose secondary input categories contain a mix of variables and concrete categories, such as

$$X/[c_{i-1}] \quad [c_{i-1}]\mathbf{1}_j \Rightarrow X\mathbf{1}_j \quad \text{if } v_j \text{ occurs in } c_i \quad (16)$$

Like the availability of composition rules of unbounded degree, the ability to use rule restrictions seems to be a very powerful feature, and one that perhaps most clearly sets VW-CCG apart from TAG. Moreover, as already mentioned, rule restrictions also play a crucial role with respect to weak generative capacity ([Kuhlmann, Koller, and Satta 2015](#)).

Note that we could replace rules of the form (16) with rules without variables; but then, for fixed values of i and j and, say, for the truth assignment $[v_j \mapsto \mathbf{1}]$, we would have to include into the grammar all of the 2^{n-1} rules of the form

$$X/[c_{i-1}] \quad [c_{i-1}]/A_1 \cdots /A_{j-1}/[v_j \mapsto \mathbf{1}]/A_{j+1} \cdots /A_n \Rightarrow X/A_1 \cdots /A_n$$

where each A_h is a concrete atomic category of the form $[v_h \mapsto 1]$ or $[v_h \mapsto 0]$. This would break our proof because reductions must use polynomial time (and space). Note also that what is crucial here is not the use of variables or the use of concrete categories in a rule's secondary input; rather, it is the combination of the two that allows us to check clauses against truth assignments.

4. Complexity With Categories for Unpronounced Words

In this section we investigate the computational complexity of the universal recognition problem for unrestricted VW-CCG, where one is allowed to assign lexicon entries even to unpronounced words. We show the following:

Theorem 2

The universal recognition problem for unrestricted VW-CCG is EXPTIME-complete.

The proof of this theorem is more involved than the proof of the NP-completeness result in Section 3. We start in Section 4.1 by introducing *alternating Turing machines*, which provide the computational framework for our proof. We then present the hardness part of the proof in Section 4.2, and the membership part in Section 4.3. We finally discuss our result in Section 4.4.

4.1 Alternating Turing Machines

The alternating Turing machine, or ATM for short, is a generalization of the well-known nondeterministic Turing machine in which there are two types of states: existential states and universal states. When the machine is in an existential state, it behaves in the standard way, accepting the input if there is at least one transition that eventually leads to an accepting state. In contrast, when the machine is in a universal state, it accepts only if *every* possible transition eventually leads to an accepting state.

For our proof we use ATMs working in polynomial space, which means that the length of the tape is bounded by a polynomial in the length of the input. This resource-restricted model is well-known in the literature, and it exactly characterizes the class of all decision problems that are solvable by a *deterministic* Turing machine working in exponential time (Chandra, Kozen, and Stockmeyer 1981), which is precisely what we need for our proof.

To simplify the notation and some of our proofs, we use ATMs that operate on a *circular tape*, and can only move their head to the right. The same model has previously been used by, among others, Jež and Okhotin (2011). It is not hard to see that, as long as we work under the restriction to polynomial space, every move to the left in the standard model can be simulated by a (polynomial) number of moves to the right in the circular tape model.

Formal Definition. Formally, an **alternating Turing machine** (for the purposes of this article) is a structure $M = (Q, \Sigma, \delta, q_0, g)$ where: Q is a finite set of **states**; Σ is an alphabet of **tape symbols**, which we assume includes the special blank symbol $\#$; $\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma)$ is the **transition relation**; $q_0 \in Q$ is the initial state; and $g: Q \rightarrow \{\exists, \forall, A, R\}$ is the type function, which assigns a type to each state. The four different types for a state

are existential (\exists), universal (\forall), accepting (A), and rejecting (R); their semantics will become clear below.

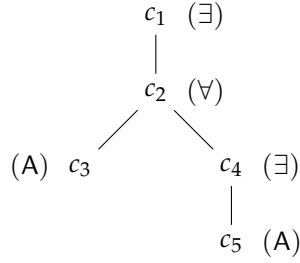
We denote transitions in δ as $(q, a) \rightarrow (q', a')$. Transitions are subject to the restriction that the state to the left of the arrow must be either existential or universal. This means that no transition is possible out of an accepting or a rejecting state; when an ATM reaches such a state, it necessarily stops. We also require that for every universal state q and tape symbol a , there are exactly two transitions with left-hand side (q, a) . This is without loss of generality: If a machine does not already have this property, then one can construct (in polynomial time) an equivalent polynomial-space ATM with circular tape satisfying it; a similar construction for general ATMs is sketched by Papadimitriou (1994, Theorem 8.2).

Configurations. Let $w \in \Sigma^*$ be an input string for w , and let $n = |w|$ and $m = p_M(|w|)$, where p_M is the machine-specific polynomial that defines the maximal tape length. A **configuration** of M relative to w is a pair $c = (q, \alpha)$, where $q \in Q$ is some state and $\alpha \in \Sigma^*$ is a sequence of tape symbols with length $|\alpha| = m$. The intended interpretation of c is that the current state of M is q , the content of the circular tape is represented by α , and the tape head is positioned to read the first symbol of α . In particular, the initial configuration of M for w , denoted by $I_M(w)$, takes the form $I_M(w) = (q_0, w\#^{m-n})$, meaning that, at the start of the computation, the machine is in the initial state, the tape consists of the n symbols of the input string w followed by $m - n$ blanks, and the tape head is positioned to read the first symbol of w . A configuration is called existential, universal, accepting or rejecting, based on the type of its state.

Successors. Let $t = (q, a) \rightarrow (q', a')$ be a transition. The intended interpretation of t is that if M is in state q and reads tape symbol a , then it overwrites a with a' , moves its tape head one cell to the right (which is always possible because the tape is circular), and continues the computation in state q' . Formally, let $c = (q, \alpha\alpha)$ be a configuration of M . The **successor** of c with respect to t , denoted by $t(c)$, is the configuration $c' = (q', \alpha a')$, where the string $\alpha a'$ encodes the fact that the symbol a has been overwritten with a' and the circular tape has been rotated one position to the right, so that the head now is positioned to read the first symbol of α . Note that, due to our restrictions on the transition relation, a universal configuration has exactly two successors.

Acceptance. We choose to represent computations of M as trees whose nodes are labeled with configurations of M , and whose edges reflect the “successor of”-relation between configurations. Formally, the set of **accepting computations** is defined recursively as follows (recall our definition of tree terms in Section 2.3):

- Every accepting configuration c forms a one-node accepting computation.
- Let c be an existential configuration and let γ be an accepting computation whose root node is labeled with some successor of c . Then $c(\gamma)$ is an accepting computation.

**Figure 8**

An accepting run. State types for each configuration c_i are indicated in parentheses.

- Let c be a universal configuration and let γ_1, γ_2 be accepting computations whose root nodes are labeled with the two successors of c . Then $c(\gamma_1, \gamma_2)$ is an accepting computation.

A sample accepting computation is shown in Figure 8. A machine M **accepts** a string w if there exists an accepting computation γ whose root node is labeled with the initial configuration $I_M(w)$. The set of all strings that are accepted by M is denoted by $L(M)$.

Characterization of EXPTIME. As already mentioned, the reason that we are interested in polynomial-space alternating Turing machines is that they exactly characterize the class of decision problems solvable in exponential time. This is expressed by the following lemma, which is basically Corollary 3.6 in [Chandra, Kozen, and Stockmeyer \(1981\)](#).

Lemma 3

The following decision problem is EXPTIME-complete: Given a polynomial-space alternating Turing machine M and a string w , is $w \in L(M)$?

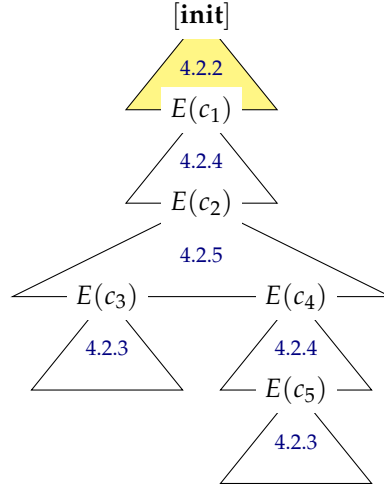
Since a polynomial-space circular-tape ATM can simulate any polynomial-space ATM at no additional asymptotic space cost, we conclude that Lemma 3 also holds for polynomial-space circular-tape ATMs. In the following we therefore use Lemma 3 as referring to polynomial space circular tape ATMs.

4.2 EXPTIME-Hardness

Let M be a polynomial-space circular-tape ATM and let w be an input string for M . In this section we show how to construct, in polynomial time and space, a VW-CCG grammar G such that $L(G) = \{\varepsilon\}$ if $w \in L(M)$, and $L(G) = \emptyset$ if $w \notin L(M)$. This means that we can test the condition $w \in L(M)$ by checking whether G generates the empty string. When combined with Lemma 3, this reduction proves the hardness part of Theorem 2:

Lemma 4

The universal recognition problem for unrestricted VW-CCG is EXPTIME-hard.

**Figure 9**

Schematic structure of the derivation in G that corresponds to the accepting computation in Figure 8. The derivation is displayed with the root node at the top to make the correspondence with Figure 8 more evident. Each $E(c_i)$ is the category that encodes the configuration c_i .

For the remainder of this section, we fix a polynomial-space circular-tape ATM $M = (Q, \Sigma, \delta, q_0, g)$ and an input string $w \in \Sigma^*$. Let p_M be the polynomial that bounds the length of the tape of M , and let $m = p_M(|w|)$.

Basic Idea. The basic idea behind our construction is straightforward: We shall set up things in such a way that the derivations of G correspond to accepting computations of M for w . To illustrate this idea, Figure 9 shows the schematic structure of a derivation that corresponds to the accepting computation in Figure 8. Note that in order to make the correspondence more evident, contrary to our previous convention, we now draw the derivation with the root node at the top. We see that the derivation is composed out of a number of smaller fragments (drawn as triangles). With the exception of the fragment at the top of the tree (which we need for technical reasons), there is one fragment per node of the accepting computation. Each fragment is labeled with a reference to the subsection of this article that describes how we set up the grammar G to derive that fragment.

One way to view our construction is that it establishes a structure-preserving map from accepting computations of M to derivations of G . This map replaces each configuration c in an accepting computation by a fragment, and continues the transformation at the subtrees below c . A fragment is like a small derivation tree, except that one or two of its leaf nodes may be labeled with (possibly complex) categories instead of lexicon entries. These nodes, which we refer to as the **distinguished** leaf nodes of the fragment, serve as slots at which the fragments that result from the recursive transformation of the subtrees can be substituted. The root node of a fragment is labeled with a category that encodes the configuration c that the fragment replaces; we denote this category by $E(c)$. More specifically, the shape of the fragment depends on the type of c :

- For every accepting configuration c , the grammar derives a fragment with no distinguished leaf nodes. The category at the root node of this fragment is $E(c)$. The lexicon entries and rules required to derive the fragment are described in Section 4.2.3.
- For every existential configuration c and for every transition t that can be applied to c , the grammar derives a fragment with a single distinguished leaf node. The category at the root node of this fragment is $E(c)$, and the category at the distinguished leaf node is $E(t(c))$, the encoding of the successor of c under t . (Section 4.2.4)
- For every universal configuration c , the grammar derives a fragment with two distinguished leaf nodes. The category at the root node of this fragment is $E(c)$, and the categories at the distinguished leaf nodes are $E(t_1(c))$ and $E(t_2(c))$, the encodings of the two successors of c . (Section 4.2.5)
- Finally, for the initial configuration $I_M(w)$, the grammar derives a fragment with a single distinguished leaf node. The category at the root node of this fragment is the distinguished category of G , and the category at the distinguished leaf node is $E(I_M(w))$. This is the highlighted fragment in Figure 9. (Section 4.2.2)

Those leaf nodes of a fragment that are not distinguished leaf nodes will always be labeled with lexicon entries for the unpronounced word, that is, entries of the form $\varepsilon := X$. Because of this, the only string that our grammar may accept is the empty string. As we will make sure that all (and only) the accepting computations of M over w receive corresponding derivations in G , this amounts to saying that G has at least one derivation if and only if $w \in L(M)$.

Technical Remark. Before we continue, we would like to make a technical remark that may make the following construction easier to understand. In the proof of Lemma 1, we constructed a grammar that produced derivations simulating a process of guessing and verifying a variable assignment for an instance of SAT. This process has a purely linear (albeit non-deterministic) structure, which is reflected in the fact that the derivation trees produced by the grammar are essentially unary-branching. For such trees, it does not make much of a difference whether we read them bottom-up (from the leaves to the root) or top-down (from the root to the leaves), and in our construction we simply adopted the former perspective, which is the conventional one for CCG.

In this proof, because of the branching nature of the computations of M , the derivation trees of the grammar G will no longer be unary-branching; and because the branching in an accepting computation of M occurs on the paths from the initial configuration to the accepting configurations, the derivation trees of the grammar G need to have the encoding of the initial configuration at the root and the encodings of the accepting configurations at the leaves. This will require us to change perspective and read the derivation trees top-down—and consequently the rules of G from the output category to the input categories. This is the opposite direction compared to what is conventional for CCG.

Derivation fragment for an initial configuration $c_I = (q_0, ab)$. Note how the symbols of c_I are collected in several steps along the highlighted path.

$$E(c) = [q] / [a_1] \cdots / [a_m]$$
$$E(t(c)) = [q'] / [a_2] \cdots / [a_m] / [a']$$

Lexicon Entries and Rules. More generally now, assume that the initial configuration for M on w is $c_I = (q_0, a_1 \cdots a_m)$, where $w = a_1 \cdots a_n$ and $a_h = \#$ for each h with $n < h \leq m$. To support fragments such as the one in Figure 10, we introduce lexicon entries $\varepsilon := [\mathbf{init}]/[q_0]$ and $\varepsilon := [a]$, where $a \in \Sigma$ is any tape symbol. We also introduce

the following rules:

$$[\mathbf{init}] \square / [a] \quad [a] \Rightarrow [\mathbf{init}] \square \quad (20)$$

$$[\mathbf{init}] \square / [q_0] \quad [q_0] / [a_1] \cdots / [a_m] \Rightarrow [\mathbf{init}] \square / [a_1] \cdots / [a_m] \quad (21)$$

A rule of the form (20) allows the application of a category with target $[\mathbf{init}]$ to any atomic category $[a]$ representing a tape symbol; this implements the nondeterministic pushing to the tape stack that we introduced above. A rule of the form (21) is a composition rule of degree m that restricts the target of the primary input category to the distinguished category $[\mathbf{init}]$, and the secondary input to the category $E(c_I)$. This implements the check in the final step of the fragment—if the category at the distinguished leaf does not encode the initial configuration at this point, then the derivation will reach a dead end.

Computational Complexity. We now deal with the computational resources required by this step of the construction. Each of the lexicon entries above is size-bounded by a constant that does not depend on $|M|$ or $|w|$. This size bound also holds for each rule of the form (20). The size of a rule of the form (21) is in $\mathcal{O}(m)$. We can then construct and store each lexical entry and each rule with time (and space) in $\mathcal{O}(m)$. Furthermore, the total number of lexicon entries and rules added to the grammar in this step is in $\mathcal{O}(|\Sigma|)$. We thus conclude that this step of the construction can be carried out in time (and space) polynomial in $|M|$ and $|w|$.

4.2.3 Accepting Configurations. Next we present lexicon entries and rules needed to terminate derivations of the accepting configurations of M . To give a concrete example, suppose that $c = (q, ab)$ is accepting, and that the grammar has already derived the category $E(c)$. Then the grammar also derives the fragment shown in Figure 11. Following the path highlighted in the figure, from the root to the leaf, the derivation first checks whether $E(c)$ indeed encodes an accepting configuration, and retargets to a special atomic category $[\mathbf{accept}]$ (22). After this, the fragment empties the tape stack, using the same type of derivation that we used to assemble the truth assignment in Figure 5.

Lexicon Entries and Rules. Let $q \in Q$ with $g(q) = A$, and let $a \in \Sigma$. We introduce the following lexicon entries:

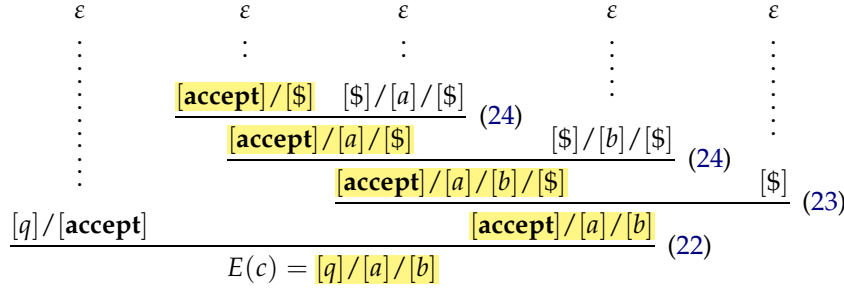
$$\varepsilon := [q] / [\mathbf{accept}] \quad \varepsilon := [\mathbf{accept}] / [\$] \quad \varepsilon := [\$] / [a] / [\$] \quad \varepsilon := [\$]$$

We also introduce the following rules:

$$[q] \square / [\mathbf{accept}] \quad [\mathbf{accept}] / X_1 \cdots / X_m \Rightarrow [q] \square / X_1 \cdots / X_m \quad (22)$$

$$[\mathbf{accept}] \square / [\$] \quad [\$] \Rightarrow [\mathbf{accept}] \square \quad (23)$$

$$[\mathbf{accept}] \square / [\$] \quad [\$] / [a] / [\$] \Rightarrow [\mathbf{accept}] \square / [a] / [\$] \quad (24)$$

**Figure 11**

Derivation fragment for an accepting configuration $c = (q, ab)$.

A rule of the form (22) is a composition rule of degree m that restricts the target of its primary input to an accepting state; this ensures that only categories encoding accepting configurations can yield subderivations of the form shown in Figure 11. In all other cases the derivation will either rewrite the configuration (for existential and universal configurations, see below), or else will reach a dead end (for rejecting configurations). The only rules that can be used after a rule of the form (22) are rules of the forms (23) and (24), which jointly implement the emptying of the tape stack that we described above.

Computational Complexity. Each of the lexicon entries above is size-bounded by a constant that does not depend on $|M|$ and $|w|$. This size bound also holds for rules of the forms (23) and (24). The sizes of rules of the form (22) are in $\mathcal{O}(m)$. The number of lexicon entries and rules that we add to G in this step is in $\mathcal{O}(|\Sigma|)$.

4.2.4 Transitions Out of Existential Configurations. We now turn to the fragments that simulate transitions out of existential configurations. Figure 12 shows how such a fragment looks like for a configuration $c = (q, ab)$ and a transition $t = (q, a) \rightarrow (q', a')$. The derivation starts by checking whether the category at the root node indeed encodes an existential configuration, and then retargets it to the transition-specific category $[t]$ (25). The derivation then extends the tape stack by the new symbol a' (26). In a last step it simultaneously discards the category for the previous tape symbol a and retargets to $[q']$ (27). After this, the category at the distinguished leaf encodes the configuration $t(c) = (q', ba')$.

Lexicon Entries and Rules. Let $q \in Q$ be any existential state, and let $t = (q, a) \rightarrow (q', a')$ be any transition out of q . We introduce the following new lexicon entries:

$$\varepsilon := [q]/[t] \quad \varepsilon := [t]/[a]/[q']$$

$$\begin{array}{c}
\begin{array}{ccc}
\varepsilon & & \varepsilon \\
\vdots & & \vdots \\
\vdots & & \vdots \\
\vdots & & \vdots
\end{array}
\quad
\begin{array}{c}
\begin{array}{c} \triangle \\ \hline \end{array} \\
\begin{array}{c} [t]/[a]/[q'] \quad E(t(c)) = [q']/[b]/[a'] \\ \hline [t]/[a]/[b]/[a'] \end{array} \quad (27)
\end{array}
\quad
\begin{array}{c}
\varepsilon \\
\vdots \\
\vdots \\
[a']
\end{array}
\end{array}
\quad (26)$$

$$\begin{array}{c}
[q]/[t] \quad \begin{array}{c} [t]/[a]/[b] \\ \hline \end{array} \quad (25) \\
\hline
E(c) = [q]/[a]/[b]
\end{array}$$

Figure 12

Fragment for a transition $t = (q, a) \rightarrow (q', a')$ out of an existential configuration $c = (q, ab)$.

We also reuse the lexicon entries $\varepsilon := [a]$ that we introduced in Section 4.2.2. Finally, we introduce the following rules:

$$[q] \square / [t] \quad [t] / X_1 \cdots / X_m \Rightarrow [q] \square / X_1 \cdots / X_m \quad (25)$$

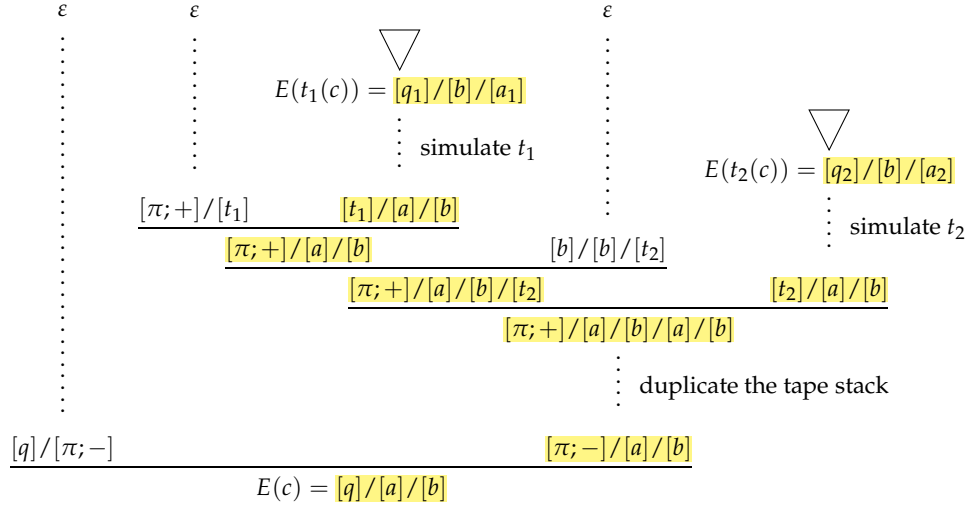
$$[t] \square / [a'] \quad [a'] \Rightarrow [t] \square \quad (26)$$

$$[t] \square / [q'] \quad [q'] / X_1 \cdots / X_m \Rightarrow [t] \square / X_1 \cdots / X_m \quad (27)$$

A rule of the form (25) is a composition rule of degree m that simultaneously restricts the target of its primary input to q and the target of its secondary input to t . A rule of the form (26) is an application rule that matches t (the target of its primary input) with the tape symbol a' produced by t (its secondary input). A rule of the form (27) is a composition rule of degree m that matches t (the target of its primary input) with the state q' resulting from the application of t .

Computational Complexity. Again, each of the above lexical entries and rules has size in $\mathcal{O}(m)$. The number of rules of the form (27) added to the grammar is bounded by the possible choices of the transition t (q' is unique, given t), and is thus a polynomial function of $|M|$. Similar analyses apply to the other rules and lexical entries. We thus conclude that the overall contribution to $|G|$ in this step is polynomial in the size of the input, and the construction can be carried out in polynomial time, too.

4.2.5 Transitions Out of Universal Configurations. We finally present the lexicon entries and rules needed to simulate transitions out of universal configurations. This is the most involved part of our construction. To give an intuition, Figure 13 provides a bird's eye view of the fragment that our grammar derives for a universal configuration $c = (q, ab)$ and a pair of transitions $\pi = (t_1, t_2)$ where $t_1 = (q, a) \rightarrow (q_1, a_1)$ and $t_2 = (q, a) \rightarrow (q_2, a_2)$. (Recall that we may assume that every universal configuration has exactly two applicable transitions.) On a high level, the derivation in the fragment proceeds in three phases as follows. First, it duplicates the tape stack of the root category $E(c)$. Second, it splits the duplicated stack into two identical halves, each targeted at one of the two transitions. Third, it simulates (in two separate branches) the two transitions on their respective halves to arrive at the two leaf categories $E(t_1(c))$ and $E(t_2(c))$. (Note

**Figure 13**

Bird's-eye view of the derivation fragment for a pair of transitions $\pi = (t_1, t_2)$ out of a universal configuration (q, ab) , where $t_1 = (q, a) \rightarrow (q_1, a_1)$ and $t_2 = (q, a) \rightarrow (q_2, a_2)$.

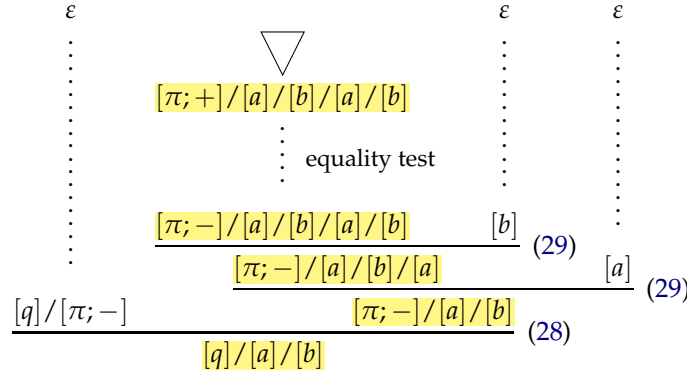
that the fragment here differs from the one in Figure 12 in that it has *two* distinguished leaves, not one.) In the following we describe the three phases in detail.

Phase 1: Duplicating the Tape Stack. We illustrate this phase of the derivation in Figure 14. The derivation starts by checking whether the root category $E(c)$ indeed encodes a universal configuration, and records this fact by retargeting to $[\pi; -]$ (28). The intended interpretation of this category is that the derivation is simulating the transition pair π but has not yet duplicated the tape stack ($-$). The derivation then nondeterministically extends the tape stack (29), in much the same way as for the initial configuration in Section 4.2.2. At the end of the phase, the derivation tests whether the two halves of the stack are equal, that is, whether the nondeterministic extension indeed created an exact copy of the initial stack. This test is crucial for our construction, and we describe it in more detail below. If the test is successful, the derivation retargets to $[\pi; +]$, signifying that the derivation has successfully duplicated the tape stack.

To support derivations such as the one in Figure 14, we introduce the following lexicon entries and rules. Let $q \in Q$ be any universal state, and let $\pi = (t_1, t_2)$ be any pair of transitions where $t_1 = (q, a) \rightarrow (q_1, a_1)$ and $t_2 = (q, a) \rightarrow (q_2, a_2)$. Let also $b \in \Sigma$ be any tape symbol. We introduce the lexicon entry $\varepsilon := [q]/[\pi; -]$ and reuse the entries of the form $\varepsilon := [b]$ that we introduced in Section 4.2.2. The following rules implement the retargeting of the root category and nondeterministic extension of the tape stack:

$$[q] \square / [\pi; -] \quad [\pi; -] / X_1 \cdots / X_m \quad \Rightarrow \quad [q] \square / X_1 \cdots / X_m \quad (28)$$

$$[\pi; -] \square / [b] \quad [b] \quad \Rightarrow \quad [\pi; -] \square \quad (29)$$

**Figure 14**

Simulation of transitions out of universal configurations, phase 1: Duplicating the tape stack.

A rule of the form (28) is a composition rule of degree m that simultaneously restricts the target of its primary input to $[q]$ and the target of its secondary input to $[\pi; -]$. A rule of the form (29) is an application rule that restricts the target of its primary input to $[\pi; -]$.

It remains to describe how to implement the equality test. To give an intuition, Figure 15 shows a derivation that implements the test for the tape ab of our example configuration. For readability, we have numbered the arguments on the tape stack. Step (30) uses a composition rule of degree $2m$ to retarget the root category to a new atomic category $[\pi; =_1]$. The intended interpretation of this category is that the derivation needs to check whether the two halves of the tape stack agree at position 1 and $m + 1$. Accordingly, the composition used in step (31) is restricted in such a way that it can only be instantiated if the two atomic categories at positions 1 and 3 are equal. Similarly, the composition used in step (32) can only be instantiated if the two categories at positions 2 and 4 are equal. It is not hard to see that this can be scaled up to m tests, each of which tests the equality of the categories at positions i and $m + i$. Taken together, these tests check whether the two halves of the tape are indeed identical.

More formally now, let $b \in \Sigma$ be any tape symbol. We introduce the following shorthand notation, where $1 \leq i \leq m$ is any tape position:

$$\eta_{i,b} \equiv /X_1 \cdots /X_{i-1} / [b] / X_{i+1} \cdots /X_m / X_{m+1} \cdots /X_{m+i-1} / [b] / X_{m+i+1} \cdots /X_{2m}$$

Thus $\eta_{i,b}$ is a sequence of $2m$ slash-variable pairs, except that X_i and X_{m+i} have been replaced with the concrete atomic category $[b]$. Then to support derivations such as the one in Figure 15, we introduce the following lexicon entries, where $1 \leq i < m$:

$$\varepsilon := [\pi; -] / [\pi; =_1] \quad \varepsilon := [\pi; =_i] / [\pi; =_{i+1}] \quad \varepsilon := [\pi; =_m] / [\pi; +]$$

$$\begin{array}{c}
\varepsilon \qquad \qquad \varepsilon \qquad \qquad \varepsilon \qquad \qquad \nabla \\
\vdots \qquad \qquad \vdots \qquad \qquad \vdots \qquad \qquad \vdots \\
\frac{[\pi; -] / [\pi; =_1] \quad \frac{[\pi; =_1] / [\pi; =_2] \quad \frac{[\pi; =_2] / [\pi; +] \quad [\pi; +] / [a]_1 / [b]_2 / [a]_3 / [b]_4}{(32)}}{[\pi; =_2] / [a]_1 / [b]_2 / [a]_3 / [b]_4} \quad (31)}{[\pi; =_1] / [a]_1 / [b]_2 / [a]_3 / [b]_4} \quad (30) \\
\frac{[\pi; -] / [a]_1 / [b]_2 / [a]_3 / [b]_4}{(30)}
\end{array}$$

Figure 15Equality test for the tape $abab$.

$$\begin{array}{c}
\varepsilon \qquad \qquad \nabla \qquad \qquad \varepsilon \\
\vdots \qquad \qquad \vdots \qquad \qquad \vdots \\
\frac{[\pi; +] / [t_1] \quad [t_1] / [a] / [b]}{(35)} \quad \frac{[b] / [b] / [t_2]}{(34)} \quad \nabla \\
\frac{[\pi; +] / [a] / [b]}{(35)} \quad \frac{[t_2] / [a] / [b]}{(33)} \\
\frac{[\pi; +] / [a] / [b] / [t_2]}{(34)} \quad \frac{[t_2] / [a] / [b]}{(33)} \\
\frac{[\pi; +] / [a] / [b] / [a] / [b]}{(33)}
\end{array}$$

Figure 16

Simulation of transitions out of universal configurations, phase 2: Splitting the tape stack.

We also introduce the following composition rules of degree $2m$, for $1 \leq i < m$:

$$[\pi; -] \square / [\pi; =_1] \quad [\pi; =_1] / X_1 \cdots / X_{2m} \Rightarrow [\pi; -] \square / X_1 \cdots / X_{2m} \quad (30)$$

$$[\pi; =_i] \square / [\pi; =_{i+1}] \quad [\pi; =_{i+1}] \eta_{i,b} \Rightarrow [\pi; =_i] \square \eta_{i,b} \quad (31)$$

$$[\pi; =_m] \square / [\pi; +] \quad [\pi; +] \eta_{m,b} \Rightarrow [\pi; =_m] \square \eta_{m,b} \quad (32)$$

Phase 2: Splitting the Tape Stack. In the second phase, the derivation branches off into two subtrees, as already illustrated in Figure 13, and repeated in Figure 16. This derivation simulates the “splitting” of the tape stack into two (identical) halves. To implement it, we introduce lexicon entries $\varepsilon := [\pi; +] / [t_1]$ and $\varepsilon := [b] / [b] / [t_2]$, where $b \in \Sigma$ is any tape symbol. We also introduce the following rules:

$$[\pi; +] \square / [t_2] \quad [t_2] / X_1 \cdots / X_m \Rightarrow [\pi; +] \square / X_1 \cdots / X_m \quad (33)$$

$$[\pi; +] \square / [b] \quad [b] / [b] / [t_2] \Rightarrow [\pi; +] \square / [b] / [t_2] \quad (34)$$

$$[\pi; +] \square / [t_1] \quad [t_1] / X_1 \cdots / X_m \Rightarrow [\pi; +] \square / X_1 \cdots / X_m \quad (35)$$

Rule of the forms (33) and (35) are composition rules of degree m . Note that this ensures that the categories targeted at $[t_1]$ and $[t_2]$ encode a tape of m symbols. A rule of the form (34) is a composition rule of degree 2.

Phase 3: Simulating the Two Transitions. In the third and final phase, the derivation simulates the two transitions t_1 and t_2 . To implement this phase we do not need to introduce any new lexicon entries or rules; we can simply reuse part of the construction that we presented in Section 4.2.4 for the existential states. More specifically, we can reuse the part of that construction that starts with a category with target $[t]$ and uses rules (26) and (27).

Computational Complexity. All of the introduced lexical entries have size bounded by some constant independent of the input size. At the same time, all of the rules introduced in the four phases above have degree bounded by $2m$. It is easy to see that we can construct each of these elements in time $\mathcal{O}(m)$. Furthermore, the number of lexical entries and rules produced in this step is bounded by a polynomial function of the input size. For instance, we add to G a number $|\Sigma| \cdot m$ of rules of types (30) and (31), since there is a single rule for each choice of a tape symbol a and index i with $1 \leq i < m$. Similar analyses can be carried out for the remaining elements. We then conclude that the overall contribution to $|G|$ in this step is polynomial in $|M|$ and $|w|$, and the construction can be carried out in the same amount of time.

4.2.6 Correctness. With all of the grammar components in place, we can now deal with the correctness of G . We argue that the sentential derivations of G exactly correspond to the accepting computations of M for w . To do this, we read G 's derivations in the canonical direction, that is, from the leaves to the root. First of all, observe that the fragments introduced in the various steps of the construction all use reserved target categories, and they all use rules with target restrictions for these categories. In this way, it is not possible in a derivation to mix fragments from different steps—that is, fragments cannot be broken apart.

Accepting Configurations. A (sentential) derivation in G starts with the fragments introduced in Section 4.2.3. Each of these fragments uses composition rules to combine several tape symbols into a category of the form $[\mathbf{accept}]_\alpha$, and then switches to a category of the form $[q]_\alpha$ with $g(q) = A$. Because of the degree restriction of Rule (22), the switch is only possible if α has exactly m arguments; in all other cases the derivation will come to a dead end, that is, it will not derive the distinguished category of the grammar. The categories $[q]_\alpha$ composed by the fragments of Section 4.2.3 encode accepting configurations of M , and it is not difficult to see that all possible accepting configurations with $g(q) = A$ can be generated by these fragments. These are exactly the leaves of our valid computation trees.

Existential Configurations. The derivation freely attempts to apply the transitions of M in reverse to the categories obtained as above and, recursively, to all the categories that result from the application of these transitions. More precisely, a transition t applying to an existential configuration is simulated (in reverse) on a category $[q]_\alpha$ using the fragment of Section 4.2.4. This is done using Rule (27), which switches from a category with target $[q]$ to a category with target $[t]$, and produces a new category whose stack has $m + 1$ arguments. At this point, only some rule of type (26) can be applied, resulting

in the reduction of the stack, immediately followed by some rule of type (25), which is a composition rule of degree m . If the derivation were to use more than one occurrence of (26), then it would derive a category whose stack contains fewer than m elements. As a consequence, Rule (25) would no longer be applicable, because of the restriction on the composition degree, and the whole derivation would come to a dead end.

Universal Configurations. The derivation can also simulate (in reverse) the two transitions t_1 and t_2 applying to a universal state q . This is done using the fragments of Section 4.2.5. In this case the derivation starts with Rules (27) and (26) used for the existential states; but now the involved categories have targets $[t_i]$ disjoint from the targets used for the existential states, since transitions t_i apply to universal states. The simulation of t_1 and t_2 results in categories $[q]\alpha$ and $[q]\alpha'$ with the same target $[q]$ and with m arguments each. These categories are then merged into a new category $[q]\alpha\alpha'$ by concatenating their stacks, and an equality test is successively carried out on $\alpha\alpha'$. If the test is successful, the derivation pops an arbitrary number of arguments from $[q]\alpha\alpha'$, resulting in a new category of the form $[q]\alpha''$. Rule (28) can then be applied only in case $[q]\alpha''$ has exactly m arguments. This means that $[q]\alpha''$ encodes one of the configurations of M , and that $\alpha = \alpha' = \alpha''$.

Initial Configuration. Finally, if the above process ever composes a category $[q_0]\alpha$ encoding the initial configuration of M relative to the input string w , then the derivation uses the fragment of Section 4.2.2. The rules of this fragment switch the target category from $[q_0]$ to **[init]**, the distinguished category of G , and then pop the stack arguments, providing thus a sentential derivation for ε .

This correctness argument finally concludes the proof of our Lemma 4.

4.3 Membership in EXPTIME

It remains to prove the following:

Lemma 5

The universal recognition for unrestricted VW-CCG is in EXPTIME.

To show this, we extend an existing recognition algorithm by Kuhlmann and Satta (2014) that takes as input a VW-CCG G with no empty categories and a string w , and decides whether $w \in L(G)$.

Complexity of the Algorithm of Kuhlmann and Satta. The algorithm of Kuhlmann and Satta (2014) is based on a special decomposition of CCG derivations into elementary pieces, adapting an idea first presented by Vijay-Shanker and Weir (1993). These elementary pieces are specially designed to satisfy two useful properties. The first property is that each elementary piece can be stored using an amount of space that does not depend on the length of w . The second property is that elementary pieces can be shared among different derivations of w under G .

The algorithm uses dynamic programming to construct and store in a multi-dimensional parsing table all possible elementary pieces pertaining to the derivations of w

under G . From such table one can directly check whether $w \in L(G)$. Despite the fact that the number of derivations of w under G can grow exponentially with the length of w , the two properties of elementary pieces reported above allow the algorithm to run in time polynomial in the length of w . However, the runtime is not bounded by a polynomial function in the size of G , as should be expected from the hardness results reported in Section 3.1.

More specifically, let \mathcal{A} be the set of all atomic categories of the input grammar G , and let \mathcal{L} be the set of all arguments occurring in the categories in G 's lexicon. Let also d be the maximum degree of a composition rule in G , let a be the maximum arity of an argument in \mathcal{L} , and let ℓ be the maximum number of arguments in the categories in G 's lexicon. We set $c_G = \max\{d + a, \ell\}$. Kuhlmann and Satta (2014) report for their algorithm a running time in $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{L}|^{2c_G} \cdot |w|^6)$.

To see that the above upper bound is an exponential function in the size of the input, observe that the quantities $|\mathcal{A}|$ and $|\mathcal{L}|$ are both bounded by $|G|$, since each category in \mathcal{A} or in \mathcal{L} must also occur in G . Furthermore, ℓ is bounded by the maximum length of a category in G 's lexicon, and thus by $|G|$. Similarly, d is bounded by the length of some secondary component in a composition rule of G , and a is bounded by the length of some category in G 's lexicon. Then $d + a$ is bounded by $|G|$ as well. Combining the previous observations, we have that c_G is bounded by $|G|$. We can then conclude that the run time of the recognition algorithm is bounded by

$$|G|^{1+2c_G} \cdot |w|^6 = 2^{\log |G| + 2|G| \log |G|} \cdot |w|^6,$$

which is an exponential function in the size of the input.

Extension of the Algorithm of Kuhlmann and Satta to VW-CCG. As already mentioned, the algorithm by Kuhlmann and Satta (2014) works for a grammar G with no empty categories. More specifically, the algorithm starts by adding to the parsing table items of the form $[X, i, i + 1]$ for each category X that is assigned by G 's lexicon to the i -th word in the input string w . Here $[X, i, i + 1]$ represents an elementary piece of derivation consisting of a tree with two nodes: a root with label X and a child node with label the i -th word of w . In order to extend the algorithm to unrestricted VW-CCG, all we need to do is add to the parsing table items of the form $[X, i, i]$ for every empty category X in G 's lexicon and for every integer i with $0 \leq i \leq |w|$. This creates new elementary pieces of derivations accounting for unpronounced words. These pieces can be combined with each other, as well as with other pieces already existing in the table, triggering the construction of derivations that involve empty categories.

The proof of the correctness of the recognition algorithm by Kuhlmann and Satta (2014) immediately extends to the new algorithm. This is so because the proof only rests on structural properties of CCG derivations, without any assumption about the fact that these derivations involve words from w or unpronounced words. Furthermore, the exponential runtime reported above still holds for the new algorithm. This is a consequence of the fact that we use the same item representation as in the original algorithm for the elementary pieces of derivations involving unpronounced words.

While the algorithms discussed above are designed for the random access machine architecture, or RAM for short, it is well-known that any algorithm working on a RAM can be computed on a deterministic Turing machine with only polynomial time overhead; see for instance (Papadimitriou 1994, Theorem 2.5). We can thus conclude that the universal recognition problem for VW-CCG can still be solved in exponential time on a deterministic Turing machine.

4.4 Discussion

In view of our proof of Theorem 2, we now come back to the question raised in Section 3.3. Specifically, we want to further investigate the features that are responsible for the additional complexity that has been ascribed to unrestricted VW-CCG. In our reduction in Section 4.2 we have used two features of the formalism that were already discussed in Section 3.3, namely the capability to define rules with restrictions on their secondary input categories, and the capability to define rules whose secondary input categories do not have any constant bound on their arity. In addition, this time we have also exploited two new features, listed below. Again, dropping anyone of these four features would break our proof.

Structural Ambiguity. Our grammar makes crucial use of structural ambiguity. More precisely, G encodes M 's configurations relative to w into its own categories. In the worst case, the set of all such categories can be as large as an exponential polynomial function of the form $c^{p_M(|w|)}$, where c is some constant dependent on M and p_M is the polynomial bounding the tape of M . We assign all of these categories to the null string ε , therefore introducing massive ambiguity in G 's derivations. Our proof of Lemma 4 would not work if we restricted ourself to the use of unambiguous grammars. More in general, it is not clear at all whether the universal recognition problem for the class VW-CCG restricted to unambiguous grammars has an efficient solution or not. Furthermore, on a par with lexical ambiguity discussed in Section 3.3, syntactic ambiguity is an essential feature in most formalisms for the modeling of natural language syntax, including those whose universal recognition problem can be parsed in polynomial time, such as TAGs. As such, this individual feature cannot be held responsible, at least in isolation, of the complexity of the recognition problem for the class VW-CCG, and in designing an efficient version of VW-CCG we are not interested in blocking structural ambiguity.

Unlexicalized Rules. We remind the reader that, broadly speaking, a lexicalized rule in a string rewriting formalism is a rule that (directly) produces some lexical token. The rule is also thought to be specialized for that token, meaning that the rule contributes to the derivation by introducing some structure representing the syntactic frame and valencies of the token itself. We start by counting the number of nodes in a valid computation of M on w . The corresponding tree can have depth given by an exponential polynomial function of the form $c^{p_M(|w|)}$, where c is some constant dependent on M . Since all nodes of the tree can be binary branching, we derive a total number of nodes $2^{c^{p_M(|w|)}}$, which is a double exponential function in the length of w .

The basic idea of our proof is to simulate valid computations of M on w through derivations of G . In particular, each fragment in a derivation of G uniquely represents some node of a valid computation. If fragments were using lexicalized rules, the strings derived by G will not have size polynomial in the length of w , and our proof would break apart. It is therefore essential in our proof that each fragment uses unlexicalized rules, that is, generates the empty string.

5. General Discussion

The computational effect of grammar structure and grammar size on the parsing problem is rather well understood for several formalisms currently used in computational linguistics, including context-free grammar and TAG. However, to the best of our knowledge, this problem has not been investigated before for VW-CCG or other versions of CCG; see for instance Kuhlmann and Satta (2014) for discussion. In this article we have shed some light on the impact of certain features of VW-CCG on the computational complexity of the parsing problem. We have shown that the universal recognition problem for VW-CCG is dramatically more complex than the corresponding problem for TAG, despite the already mentioned weak equivalence between these two formalisms. Our results therefore solve an important open problem for VW-CCG and, more in general, open the way to techniques for analyzing the computational complexity of contemporary incarnations of CCG.

The three features of VW-CCG that are at the core of the complexity results in this article are the ε -entries, the rule restrictions, and the unbounded degree of composition rules. As already mentioned, dropping any one of these three features would break our specific constructions, invalidating our results. At the same time, it is also important here to consider the problem from the dual perspective: Currently, we do not know whether dropping *any combination* of these three features from VW-CCG would allow a polynomial time parsing algorithm for the resulting class of grammars. For instance, the recognition algorithm by Kuhlmann and Satta (2014), which has been briefly described in Section 4.3, takes as input an ε -free VW-CCG G and a string w , and decides whether $w \in L(G)$. Even if G has no rule restrictions and the degree of its composition rules is considered as a constant, the algorithm would still take exponential time in the grammar size. This shows that our understanding of the computational properties of VW-CCG, and CCG in general, is still quite limited.

Succinctness. As already mentioned VW-CCG is known to be generatively equivalent to TAG, in the weak sense, as shown by Vijay-Shanker and Weir (1994). Schabes (1990) reports that the universal recognition problem for TAG can be decided in time $\mathcal{O}(|G|^2|w|^6)$, where $|G|$ is the size of the input grammar G and $|w|$ is the length of the input sentence w . One could hope then to efficiently solve the universal recognition problem for VW-CCG by translating an input VW-CCG G into an equivalent TAG G' , and then applying to G' and the input string any standard recognition method for the latter class. However, the part of the equivalence proof by Vijay-Shanker and Weir (1994) showing how to translate VW-CCG to TAG requires the instantiation of a number

of elementary trees in G' that is exponential in $|G|$. (Trees are the elementary objects encoding the rules in a TAG.)

The fact that the same class of languages can be generated by grammar formalisms with substantially different parsing complexity naturally leads us to the notion of the *succinctness* of a grammar. In formal language theory, grammar succinctness is used to measure the expressive capacity of a grammar formalism, as opposed to its generative capacity. More precisely, grammar succinctness measures the amount of resources that different grammar formalisms put in place in order to generate the same language class. As a simple example, it is well known that certain finite languages can be generated by context-free grammars that are very compact, that is, small in size, while the same languages require finite state automata of size exponentially larger. In computational linguistics, succinctness was first discussed in the context of the formalism of ID/LP-grammar, a variant of context-free grammar where the ordering of the nonterminals in the right-hand side of a rule can be relaxed. [Moshier and Rounds \(1987\)](#) showed that ID/LP-grammars are exponentially more succinct than context-free grammars. As in the example above, this means that there are languages for which any context-free grammar must necessarily be at least super-polynomially larger than the smallest ID/LP-grammar. A similar fact holds for VW-CCG: By our result in Section 3 there are languages for which there exist small VW-CCGs but where the weakly equivalent TAG must necessarily be at least exponentially larger (unless $\text{PTIME} = \text{NP}$). If we allow ε -entries, then Section 4 provides a stronger result: We can get rid of the qualification “unless $\text{PTIME} = \text{NP}$ ”. Since we can also translate any TAG into an equivalent VW-CCG without blowing up the size of the grammar, following the construction by [Vijay-Shanker and Weir \(1994\)](#), we conclude that VW-CCG is more succinct than TAG. However, the price we have to pay for this gain in expressivity is the extra complexity in parsing for VW-CCG that we have found in this article.

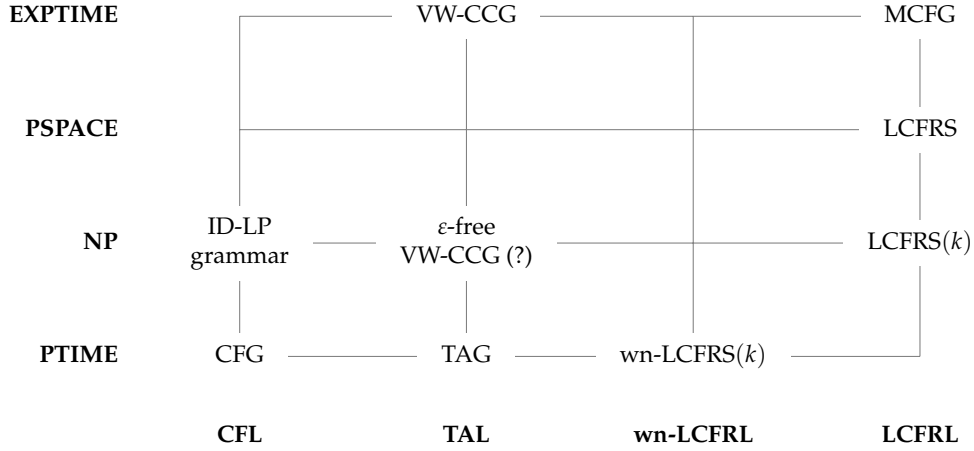
Epsilon Entries. One important issue that needs further discussion here is the role of ε -entries in VW-CCG. From the linguistic perspective, ε -entries violate one of the central principles of CCG, the Principle of Adjacency, by which combinatory rules may only apply to finitely many phonologically realized and string-adjacent entities ([Steedman 2000](#), p. 54). From the computational perspective, ε -entries represent the boundary between the results in Section 3 and Section 4. However, since we do not know whether the classes NP and EXPTIME can be separated, we cannot draw any precise conclusion about the role of ε -entries in the parsing problem. Even under the generative perspective, we do not know the exact role of ε -entries. More precisely, the proof of the weak equivalence between VW-CCG and TAG provided by [Vijay-Shanker and Weir \(1994\)](#) makes crucial use of ε -entries, and it is currently unknown whether the generative capacity of VW-CCG without ε -entries is still the same as that of TAG, or if it is strictly smaller. This is an other important open problem that attests our lack of theoretical understanding of VW-CCG. The only thing that we can say at this time is that, under the hypothesis that $\text{NP} \neq \text{EXPTIME}$, and under the hypothesis that VW-CCG without ε -entries is still generatively equivalent to TAG, the removal of ε -entries from VW-CCG would be computationally expensive.

Unbounded Composition. A second issue that we would like to discuss is related to the notion of degree of composition rules in VW-CCG. According to the original definition of VW-CCG, each *individual* grammar in this class has a specific bound on the degree of its composition rules, but there is no constant bound holding for all grammars. As already discussed in Sections 3.3 and 4.4, the complexity results in this article do exploit this property in a crucial way. However, there are two alternative scenarios that we want to consider here. In a first scenario, one could state that there exists some language-independent constant that bounds the degree of composition rules *for all* grammars in the class VW-CCG. This would break all of the constructions in this article.

The second possible scenario is one that has been briefly mentioned by Weir and Joshi (1988, Section 5.2). We could define a formalism alternative to VW-CCG, in which an individual grammar is allowed to use composition rules of unbounded degree. This would mean that the box notation introduced in Equation (8) must be used in the primary category as well as in the secondary category of a composition rule. Such move would go in the opposite direction with respect to the first scenario above, reaching the power of Turing machines (TM), as informally explained in what follows. Recall that in Section 4 we have used VW-CCG derivations to simulate moves of an ATM working with a circular tape whose size is bounded by some polynomial function of the length of the input. Specifically, we have encoded such tape into some category X , and have used X as a primary or as a secondary input category in composition rules, in order to simulate the moves of the ATM. If we now allow the use of composition rules of arbitrarily large degree within an individual grammar, we can then simulate the moves of a general TM, in a way very similar to what we have done with our ATMs. This shows that the degree of composition rules can play a very powerful role in the definition of CCG formalisms.

The Landscape of Mildly Context-Sensitive Grammars. Finally, we would like to conclude our discussion by placing our results for VW-CCG in the broader scenario of the class of mildly context-sensitive formalisms, which we have already mentioned in passing in the introduction. This provides a more complete picture of this class than what we had before. The (informal) class of mildly context-sensitive grammar formalisms had originally been proposed by Joshi (1985) to provide adequate descriptions of the syntactic structure of natural language. This class includes formalisms whose generative power is only slightly more powerful than context-free grammars, that is, far below the one of context-sensitive grammars.

In Figure 17 we map several known mildly context-sensitive formalisms into a two-dimensional grid defined by the generative capacity of the formalism (horizontal axis) and the computational complexity of the universal recognition problem (vertical axis). For comparison, we also include in the picture some formalisms generating the context-free languages. We thus start at the leftmost column of the grid with the class of context-free grammar and the class of ID-LP grammar. As already mentioned, while these two classes are generatively equivalent, ID-LP grammar is more succinct than context-free grammar and, as a consequence, the two classes do not have the same computational complexity. On the next column to the right, we reach the generative power of tree-adjoining languages, which is strictly larger than the one of context-free

**Figure 17**

Weak generative capacity (horizontal axis) versus computational complexity (vertical axis) of various mildly context-sensitive grammar formalisms.

languages. Both TAG and VW-CCG are in this column but, by the results in this article, the computational complexity of VW-CCG is far above the one of TAG, again due to the increase in expressivity for the latter class. We have also placed ϵ -free VW-CCG in this column, although we do not know at this time whether the generative capacity of this class is the same as that of general VW-CCG, hence our question mark in the figure. Next column to the right we find the class of well-nested linear context-free rewriting system with fan-out bounded by k , written $\text{wn-LCFRS}(k)$. A rewriting system in $\text{wn-LCFRS}(k)$ generates languages of string tuples, where the number of components in each tuple is bounded by a constant k called the fan-out of the system. The system exploits rules that work by combining tuple components in a way that satisfies the so-called *well-nestedness* condition, a generalization of the standard condition on balanced brackets. This mildly context-sensitive formalism has proven effective in modelling phrase structure treebanks with discontinuous constituents, as well as non-projective dependency treebanks. While this class further extends the generative capacity of TAG (as a special case, the class $\text{wn-LCFRS}(2)$ is generatively equivalent to TAG), it manages to keep the complexity of the universal recognition problem in PTIME, as shown by Gómez-Rodríguez, Kuhlmann, and Satta (2010). In the last column of our grid we have placed the class of linear context-free rewriting system (LCFRS) and the class of LCFRS with fan-out bounded by a constant k ($\text{LCFRS}(k)$), which have been originally defined by Vijay-Shanker, Weir, and Joshi (1987). Historically, LCFRS has been introduced before $\text{wn-LCFRS}-k$, and the latter class was investigated as a restricted version of LCFRS. In this column we also find the class of multiple context-free grammar (MCFG) defined by Seki et al. (1991), who also prove the generative equivalence result with LCFRS. The computational complexity results displayed in this column are from (Kaji et al. 1992) and (Satta 1992). All these systems generate string tuples but they do not satisfy the well-nestedness condition of $\text{wn-LCFRS}-k$. As a result, even in case of the class $\text{LCFRS}(k)$

where the fan-out is bounded by a constant k , these systems cannot be parsed in polynomial time, unless $\text{PTIME} = \text{NP}$, in contrast with the class $\text{wn-LCFRS}(k)$.

References

- Baldrige, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorical Grammar*. Ph.D. thesis, University of Edinburgh, Edinburgh, UK.
- Chandra, Ashok K., Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.
- Clark, Stephen and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.
- Gómez-Rodríguez, Carlos, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 276–284, Los Angeles, USA.
- Hockenmaier, Julia and Mark Steedman. 2007. CCGbank. A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33:355–396.
- Jež, Arthur and Alexander Okhotin. 2011. Complexity of equations over sets of natural numbers. *Theory of Computing Systems*, 48(2):319–342.
- Joshi, Aravind K. 1985. Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, pages 206–250.
- Joshi, Aravind K. and Yves Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer, pages 69–123.
- Kaji, Yuichi, Ryuichi Nakanishi, Hiroyuki Seki, and Tadao Kasami. 1992. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Information and Systems*, E75-D(1):78–88.
- Kuhlmann, Marco, Alexander Koller, and Giorgio Satta. 2015. Lexicalization and generative power in CCG. *Computational Linguistics*, 41(2):187–219.
- Kuhlmann, Marco and Giorgio Satta. 2014. A new parsing algorithm for Combinatory Categorical Grammar. *Transactions of the Association for Computational Linguistics*, 2(Oct):405–418.
- Lee, Kenton, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural ccg parsing with optimality guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376, Association for Computational Linguistics.
- Lewis, Mike and Mark Steedman. 2013. Unsupervised induction of cross-lingual semantic relations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 681–692, Association for Computational Linguistics.
- Lewis, Mike and Mark Steedman. 2014. A* ccg parsing with a supertag-factored model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Moshier, M. Drew and William C. Rounds. 1987. On the succinctness properties of unordered context-free grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 112–116, Stanford, CA, USA.
- Papadimitriou, Christos H. 1994. *Computational Complexity*. Addison-Wesley.
- Satta, Giorgio. 1992. Recognition of linear context-free rewriting systems. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 89–95, Newark, DE, USA.
- Schabes, Yves. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, USA.
- Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On Multiple Context-Free Grammars. *Theoretical Computer Science*, 88(2):191–229.
- Steedman, Mark. 2000. *The Syntactic Process*. MIT Press.
- Steedman, Mark and Jason Baldrige. 2011. Combinatory Categorical Grammar. In Robert D. Borsley and Kersti Börsjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Blackwell, chapter 5, pages 181–224.

- Vijay-Shanker, K. and David J. Weir. 1993.
Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.
- Vijay-Shanker, K. and David J. Weir. 1994.
The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546.
- Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 104–111, Stanford, CA, USA.
- Weir, David J. and Aravind K. Joshi. 1988.
Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 278–285, Buffalo, USA.
- Zhang, Yue and Stephen Clark. 2011.
Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 683–692, Portland, OR, USA.

